Computer Networks 54 (2010) 112–132

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

A loss-event driven scalable fluid simulation method for high-speed networks

Suman Kumar [a], Seung-Jong Park [a,*], S. Sitharama Iyengar [b]

[a] Computer Science and Center for Computation & Technology, Louisiana State University, Baton Rouge, LA 70803, United States
[b] Computer Science, Louisiana State University, Baton Rouge, LA 70803, United States

A R T I C L E I N F O

Article history:
Received 8 July 2009
Accepted 25 August 2009
Available online 1 September 2009
Responsible Editor: Prof. E. Ekici

Keywords:
Congestion control
Transport protocol
High-speed networks
Network simulation
Network modeling

A B S T R A C T

Increase of size and bandwidth of computer network posed a research challenge to evaluate proposed TCP/IP protocol and corresponding queuing policies in this scenario. Simulation provides an easier and cheaper method to evaluate TCP proposals and queuing disciplines as compared to experiment with real hardware. In this paper, problem associated with scalability of current simulation method for high-speed network case is discussed. Hence, we present a scalable time-adaptive numerical simulation driven by loss events to represent dynamics of high-speed networks using fluid-based models. The new method uses a loss event to dynamically adjust the size of a time step for a numerical solver which solves a system of differential equations representing dynamics of protocols and nodes' behaviors. A numerical analysis of the proposed protocol is discussed. A simple simulation of high-speed TCP variants is presented using our method. The simulation results and analysis show that the time-adaptive method reduces computational time while achieving the same accuracy compared to that of a fixed step-size method.

© 2009 Elsevier B.V. All rights reserved.

1. Introduction

Recently, size and bandwidth of computer networks has seen tremendous growth. Introduction of opto-electronic technology enabled high-speed links for the network. With this growth, there has been a demand for fair and efficient protocol to exploit these improvements in high-speed computer networks, such as NLR (National Lambda Rail) [1], LONI (Louisiana Optical Networks Initiative) [2], etc. Some protocol solutions have been proposed to satisfy the high-speed requirement by researchers namely Fast TCP [3], HSTCP [4], S-TCP [5], BIC-TCP [6], HAMILTON-TCP [7] on high speed networks. Development of these high-speed protocols posed a challenge to evaluate these protocols in a variety of environments. Network simulation is well accepted and widely used and a much simpler way of performance evaluation of different protocols. Also, it is well known that packet-based simulators like NS2 [8] and Opnet [9] cannot be used in the case of large simulations [10] because of its inherent bottlenecks in terms of message overhead and cpu execution time. Among the main research direction in the simulation of TCP/IP networks, we would quote parallel simulation projects, such as SSF-net [11] and emulation projects such as NistNet [12]. The requirement of cpu-time and memory increases with the increase of bandwidth as well as the size of the network. As we show, NS2 simulator is inadequate in terms of computational complexity and resources and thus we need new simulators to be modeled in a different way to deal with these problems.

Fluid simulation (through approximation of fluid dynamics) came up with an alternative way for the simulation of TCP/IP networks [13]. The fluid level simulation is based on a path wise description of the dynamics of the interaction between flows and it takes into account discrete event phenomena that are of central importance for drop-tail queues at routers and links, such as

* Corresponding author. Tel.: +1 2255782209; fax: +1 2255781465.
E-mail addresses: sumank@cct.lsu.edu (S. Kumar), sjpark@cct.lsu.edu, trysjp@hotmail.com (S.-J. Park), iyengar@csc.lsu.edu (S.S. Iyengar).

1389-1286/$ - see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.comnet.2009.08.018

congestion epochs, losses, synchronization of sources, etc. It also allows us to analyze congestion window behavior and queue dynamics. Although it satisfies the simulation of large number of flows with low bandwidth, the current method is still far behind the high-speed networks which has fewer number of flows. For the AQM (Active Queue Management) policies, the packet delay corresponding to each packet is very much dependent on the bottleneck queue attached to that link. For example, a single packet in the case of 1 Mbps bandwidth has a queue delay of 8 ms irrespective of the queuing management policies. As the bandwidth increases to 100 Mbps, this queuing delay is decreased to 0.08 ms. When dealing with this queuing delay for each packet, we have to decrease the time step for the evaluation of protocol behavior. For example to record the packet delay and the corresponding packet drop for a short period of time in TCP, we have to decrease the time step which in turn increases the execution time of the entire simulation. In the previous effort, researchers have tried to use the topology to improve the simulation time. In [13], only congested queues have been simulated with the help of network topology information. However, the presented optimization works for the large number of flows and not scalable to large bandwidths. In the case of high-speed networks (of the order of Gbps), an ideal scenario includes fewer number of high-speed flows. To find out the queue behavior in such a scenario, we need to exploit the system behavior along with the topology. It is identified that the highs peed network with optical fiber has very low random loss rates of the order of 10^{-7} [14]. In the current work, we present a loss-event driven fluid simulation method which can scale to very high bandwidths.

This paper is organized as follows: We give a brief overview of related work in this area in Section 2. In Section 3, we discuss the motivation and challenges in this area. In the same section, we discuss a simple system model and idea behind this work and introduces a general framework. Section 4 give the performance results and the validity of the proposed fluid simulator. A numerical analysis is also presented in the same section. In Section 5, we compare different high-speed transport protocols for some basic configurations using fluid simulation. Section 6 presents the conclusion and the possible future research direction in this area.

2. Related works

The aim of the fluid model is to estimate the congestion window behavior obtained by each individual flow under the competition rules imposed by TCP. These rules can be imposed from the sole knowledge of the route, the RTT of each flow, the characteristics of each router, and the link characteristics (buffer size, link capacity, scheduling, etc.) in the network. Another advantage of fluid-based approach is that the congestion control is explained as a distributed algorithm toward solving a global optimization of allocation of resources satisfying a certain fairness criterion. The cost functions are chosen such that a set of fairness criterion is maintained [15,16].

Misra et al. [17] developed a methodology to model the TCP AIMD algorithm and obtained the expected transient behavior of networks with Active Queue Management routers supporting TCP flows. They used jump process driven Stochastic Differential Equations (SDEs) to model the interaction of a set of TCP flows and Active Queue Management routers in a network setting. The derived SDEs are transformed into a set of Ordinary Differential Equations (ODEs), which can be easily solved numerically. Their formulation enables to spot a possible problem with the RED averaging mechanism related to the TCP AIMD algorithm. The formulation presented in this work is quite simple and helpful in formulating the model for other TCP variants and also to analyze the other AQM mechanisms.

The further extension of this work is done by Liu et al. [13]. In their paper, the solution techniques have been presented to reduce the simulation time by simulating and solving only the queuing equations for potential congested links. Therefore, the computation time has greatly reduced. Although the solution methodology scales well to a large number of flows, no attempt has been made for reducing the computation time for the high-speed network case. They exploited the topology information but no stress has been given to the behavior of the system (loss or protocol behavior) for the reduction of the execution time of computation/simulation which is explored in the present work.

In [18], Baccelli and Hong proposed a simplified representation of interaction of TCP flows via coupled evolution equations for simulation of large IP networks at flow level. The basis of this approach is the joint evolution of the congestion window size of long-lived (FTP type) flows controlled by TCP and sharing a single drop-tail router in the network. The modeling is done in terms of sending rate of the source, giving the instantaneous throughput fluctuations at any point of time. The important aspect of synchronization rate has been explored effectively giving more realistic simulation results as compared to previous packet-level simulators. The results obtained by this flow level simulator take into account key packet-level phenomena such as the reaction delay, the scheduling and the buffer overflows, via estimated synchronization rate.

In [19], time-driven fluid simulation is proposed to simulate high-speed networks. Here, the network elements are modeled as fluid servers where, the traffic sources can be arbitrary, including a discrete event and fluid source. Furthermore, usefulness of the fluid simulation with packet simulator has been explored in [19] where a hybrid method is used. Fluid models are used to represent aggregations of flows for which less details are required and packet-level models are used to represent the individual flows for which more details are needed.

3. A scalable method for high-speed network simulation

3.1. Problem definition and motivation

Nowadays, there have been several research initiatives which develop and deploy high-speed networks over

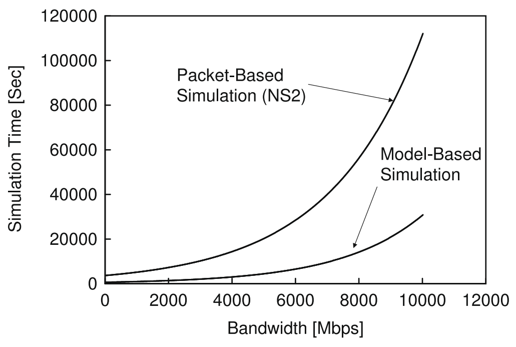


Fig. 1. Scalability as a function of network bandwidth: packet-based simulation result from NS2 simulator and model-based simulation result from a fluid level simulator using a numerical method, such as Euler method.

major research institutions. These networks, such as NLR (National Lambda Rail) [1], LONI (Louisiana Optical Networks Initiative) [2], etc. have bandwidths greater than 10 Gbps. By the virtue of DWDM (Dense Wavelength Division Multiplexing) technology [21] at link layer, a pair of optical fiber can transmit 30–40 simultaneous light waves. Therefore, the observed transmission bandwidth is 10 Gbps so that we can achieve up to 400 Gbps of bandwidth at each physical link. To catch up with those large bandwidths at the physical layer, many researchers have developed protocols at higher layers, such as transport and network layers.

To develop those protocols, it is necessary to simulate the behavior of networks in order to evaluate the performance before implementing and deploying networks. Until now, development of small or mid-scale networks of which bandwidth is less than 1 Gbps has been supported by a packet-based simulation which emulates detailed behaviors of packets or queuing theory. However, as the scale of networks increases, the execution time of the packet-based simulation methods increases exponentially due to large number of packets to process. To overcome the scalability problem, we need to develop a new model-based simulation framework which takes less amount of simulation time and uses parallel computation. For the model-based simulation, we can use an approach based on fluid dynamics which represents a behavior of an individual flow in networks.

Fig. 1 shows the execution time between the two simulation methods as a function of network bandwidth. The packet-based simulation experiences an exponential increase in its execution time as its bandwidth expands more than 1 Gbps. However, in a fluid-based simulator, for bandwidths less than 10 Gbps, the execution time is still reasonable.

Therefore, in this paper, we develop a fluid-based simulation method which predicts the behavior of transport and routing protocols over high-speed networks within less amount of execution time.

3.2. Challenge

As shown in Fig. 1, the execution time of fluid-based simulation is reasonably below that of packet-based simulation. However, its execution time is still more than several hours. Furthermore, the execution time will increase as more number of flows are getting involved in simulation. Therefore, it is necessary to develop a new fluid-based simulator which is scalable to the size of networks as well as network bandwidth.

In general, the fluid-based simulation method solves a set of Ordinary Differential Equations (ODE) which represent the dynamic behaviors of flows in networks using numerical methods, such as Euler method and Runge–Kutta method [20]. When the numerical methods solve a set of ODE, they use a time-step-size, h, which is a step-size of a solution for Eq. (1)

$$\frac{\mathrm{d}y(t)}{\mathrm{d}t} = f(t, y(t)). \tag{1}$$

To obtain a numerical estimate y_{k+1} of the Euler method, we use following equation:

$$y_{k+1} = y_k + h * f(t, y_n). \tag{2}$$

The numerical solver for a set of ODE is a time-stepped fluid-based network simulator. The accuracy of a solution is determined by the time step-size, h, and the network bandwidth because higher bandwidth creates more finer events in terms of time, such as packet departures and arrivals at nodes. For example, to represent the behavior of flows going through a link below 100 Mbps which roughly transmits 10^4 packets/s,[1] 10^{-4} s of time-step-size is a minimum time step to catch interesting events, such as packet arrival and departure, with no loss of information in the numerical sense. However, in case of 10 Gbps bandwidth networks, the step-size should be 10^{-6} s to solve equations without any loss. Therefore, simulation of high-speed networks requires a shorter time-step-size as the bandwidth of high-speed networks increases.

Clearly, for a given link capacity say C_l, the queue service time for each packet t_{st} is given as below

$$t_{st} = \frac{1}{C_l}. \tag{3}$$

As the numerical simulations have a shorter time-step-size, the total number of time steps increases. Since the computational time of the fluid-based simulation is proportional to the number of time steps, the execution time of the fluid-based simulation for high-speed networks (e.g., more than 10 Gbps bandwidth) is more than hundreds of thousand seconds (see Fig. 1).

To reduce computational time for the fluid-based simulation, we present a time-adaptive method which adjusts time-step-size based on the dynamics of flows so that it can reduce the total time steps while achieving similar level of error.

For example, the Euler method has accumulated errors, such as $e_{k+1} = y_{k+1} - y(t_{k+1})$ and $e_k = y_k - y(t_k)$, can be represented as

$$e_{k+1} = e_k + h_k(f(t_k, y_k) - f(t_k, y(t_k))) - O\left(h_k^2\right), \tag{4}$$

[1] For the convenience of calculation, we assume that the size of packet is 10,000 bits.

where y_{k+1} is an estimate and $y(t_k)$ is an exact solution $(t_{k+1} = t_k + h)$ [20].

Therefore, the accumulated error is dependent on propagational error e_k and the time-step-size h_k at each time step k. This paper will propose a new algorithm which changes the step-size h_k based on the error sensitivity while maintaining similar level of error.

For the time-adaptive Euler Method, the time-step-size h is not constant. If we take $t = t_k$ and the step-size $h_k = \Delta t_k$, we obtain from the assumption $y' = f(t_k, y(t_k))$

$$y(t_{k+1}) = y(t_k) + h_k f(t_k, y(t_k)) + O\left(h_k^2\right). \tag{5}$$

Then the global errors $e_{k+1} = y_{k+1} - y(t_{k+1})$ and $e_k = y_k - y(t_k)$ is obtained by

$$e_{k+1} = e_k + h_k(f(t_k, y_k) - f(t_k, y(t_k))) - O\left(h_k^2\right). \tag{6}$$

Therefore, the global error $e_{k+1} = y_{k+1} - y(t_{k+1})$ depends h_k and all previous errors e_i $(i = 0, \ldots, k)$. Therefore the error of kth step with time-step-size h_k $(k = 0, 1, \ldots)$ affects e_{k+1}. However, in our model, the effect of local error from larger time-step-size is relatively less than that of adaptive time. We will compare the error between constant time-step Euler methods and time-adaptive Euler methods.

Ideally, the higher order ODE solvers, such as Runge–Kutta method, might increase the order of convergence. For the convenience of calculation, this paper will use the Euler method as a basic method. However, the proposed time-adaptive method is still applicable to any higher order ODE solver.

However, the higher order methods might not be applied directly because our original network models are based on their discrete behaviors. For higher order methods, we have tested several alternatives to find the best approximation for the intermediate data needed for the methods. We have continued to find optimal approach to implement higher order methods.

3.3. Fluid model

In this section, we introduce a system model[2] where we present the fluid model and the basic idea of the algorithm used for high-speed networks with few number of flows. The network is modeled as a directed graph $G = (V, E)$ where V denotes the set of nodes and E is set of links connecting those nodes. Each link in E is served at a rate of C_l bps. Each link is associated with an AQM policy which is characterized by a packet drop probability $p_l(t)$. All the links are associated with a propagation delay for which the traffic departing from the queue associated with l arrives at the next queue after the propagation delay associate with that link. Modeling of Advance Queue Management policy is done in such a way that a packet drop takes place whenever the queue size exceeds the drop threshold with probability 1 (drop-tail behavior in our case). All the flows experience the delay which can be given by the summation of the propagation and the link delay from the source to the destination associated with their paths.

[2] The basis of current work is the fluid Simulation framework given in [13], which accounts for shaping of the flow as they traverse through different links in the network.

Without loss of generality, some of the frequently used notations in their generic forms are listed here for easy reference:

- F_i = A set of ordered queues traversed by the ith flow in forward manner.
- R_i = A set of ordered queues traversed by the ith flow in backward manner (for the acknowledgment from destination).
- $W_i(t)$ = Congestion window for ith flow at time t.
- $R_i(t)$ = Round trip time for ith flow at time t.
- M_i = Maximum congestion window limit for ith flow.
- $\lambda_i(t)$ = Loss indication rate for ith flow at time t.
- $q_l(t)$ = Queue size associated with lth link at time t.
- $p_l(t)$ = Packet drop probability at lth queue at time t.
- C_l = Service capacity/bandwidth for lth link.
- a_l = Propagation delay associated with lth link.
- $q_l^{\max}$ = Maximum queue size associated with lth link.
- n_l = Denotes number of flows traversing lth link.
- $A_l^i(t)$ = Arrival rate of ith flow at lth link at time t.

Basic equations in the form of linear and ordinary differential equations governing the flow level behavior of the network are summarized as below:

Window size:

$$\frac{dW_i(t)}{dt} = \frac{1(W_i(t) < M_i)}{R_i(t)} - \frac{W_i(t)}{2}\lambda_i(t), \tag{7}$$

where $1(W_i(t) < M_i)$ is indicator function, which has binary output. If the argument is True its value is '1' and '0' otherwise. It is associated with the window function to limit the congestion window size from going beyond the maximum allowed value.

Queue size:

$$\frac{dq_l(t)}{dt} = -1(q_l(t) > 0)C_l + \sum_{i=1}^{n_l} A_l^i(t). \tag{8}$$

Similarly queue size is restricted by the indicator function $(q_l(t) > 0)$ and can have only positive value.

Round trip time:

$$R_i(t) = \sum_{l \in F_i U R_i} a_l + \frac{q_l(t)}{C_l}. \tag{9}$$

Loss indication rate:

$$\lambda_i(t) = \sum_{l \in F_i} A_l^i(t) p_l(t). \tag{10}$$

Packet drop probability (for drop-tail queue):

$$p_l(t) = \begin{cases} 0, & q_l(t) < q_l^{\max}, \\ 1, & q_l(t) > q_l^{\max}. \end{cases} \tag{11}$$

The above equations can be used to represent the entire system as feedback system, where the loss event is used as a feedback mechanism and consecutively the AIMD adapts the congestion window to avoid the loss event by decreasing its size.

3.4. Main idea

The model introduced in the last subsection is used to give network statistics for the network equipped with drop-tail queue and TCP AIMD congestion window algorithm at the transport layer.

Figs. 2 and 3 show congestion window behavior and loss indication rate, $\lambda_i(t)$ as a function of time. Our algorithm is based on the fact that the congestion window value changes with loss indication rate. As we observe from Fig. 2, the additive increment continues till the queue overflows and a loss event occurs as indicated by loss indicator function λ as shown in Fig. 3. The queue overflow causes packet loss at that queue which causes multiplicative decrease by a factor of $\lambda/2$ (loss arrival rate) as the packet arrival rate is very high during that time. This can be easily understood by Eqs. (10) and (7). We observe that before the loss event, the window increases linearly and hence a linear solution is preferred, whereas at the time of loss event the congestion window follows a non-linear behavior. In the case of high-speed network, the loss event lasts for a very small time as compared to linear increase time, which is normal for a high bandwidth and large delay (Eq. (9)) network. Congestion window takes fairly long time (the increment part of Eq. (7)) to reach to the peak value where it overshoots causing a loss event on the queue as compared to the time while loss event lasts. When loss event happens, the window value decreases by a factor of $\frac{W}{2}\lambda$ (multiplicative decrease part in Eq. (7)). The queue is cleared immediately after the loss event with the service rate C_l as in Eq. (8) and remains empty until the window reaches the peak value again. In order to effectively record this fast decrease and recovery of congestion window from the loss event, we must have microscopic behavior of the network. Since this process lasts for a very short time, a smaller time step is preferred as compared to the time where there is no loss event and solution is more predictable. The usual method uses constant time step value and as discussed in Section 2, this constant value should be low enough to observe the network behavior for the high-speed network. However, small time step increases the overall time for simulation. For that reason, we propose a higher time step value for probing phase and smaller time step to effectively catch the loss event related characteristics in the network.

Based on the above discussion, we propose two time steps for network simulation based on loss events in the networks.

3.5. Framework

In this section, the pseudo-code of the algorithm is explained in Fig. 5 and the corresponding flow chart of the algorithm has been introduced in Fig. 4.

In the above Fig. 5, we have described the algorithm. First the value of $dt_{\min}$ is calculated by using FindDtMin( )

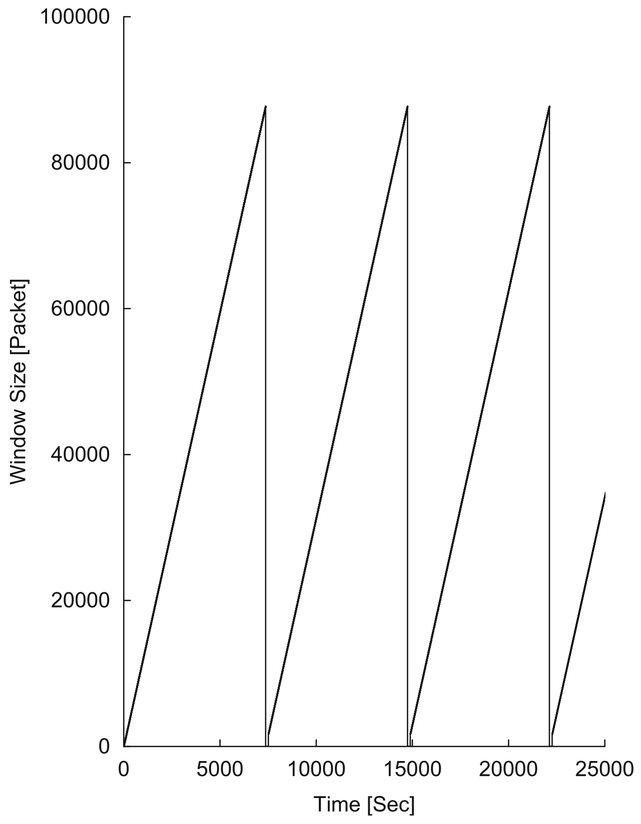


Fig. 2. Congestion window behavior at high-speed optical network and 70 ms delay.

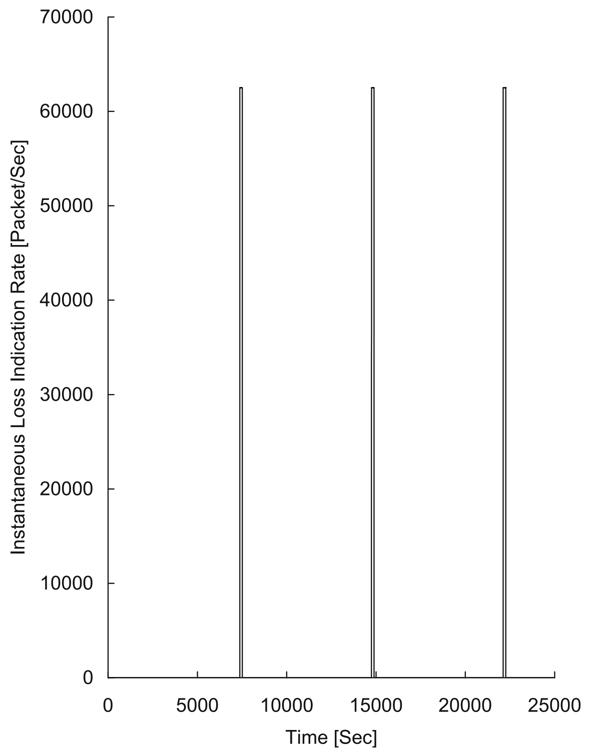
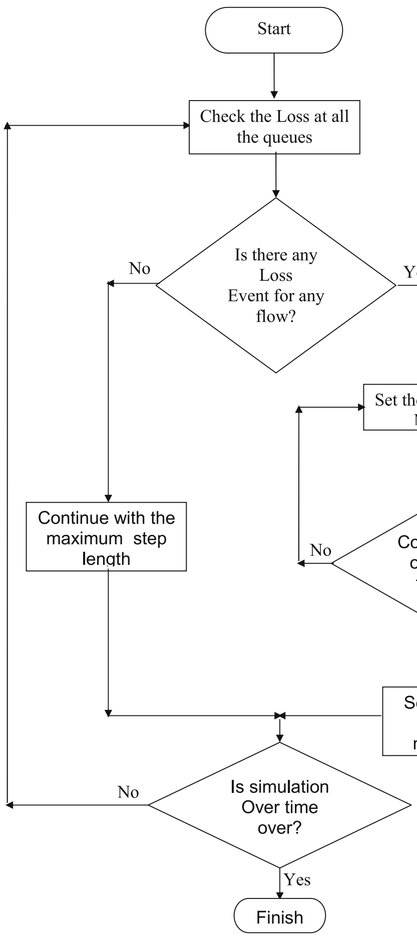


Fig. 3. Loss indication rate at high-speed optical network, 10 Gbps and 70 ms delay.

function which finds the minimum time step for simulation, as described later in this section. $dt_{\max}$ is also set as an input parameter. Further the decision as to which one out of two values should be used is made based on loss information. Since TCP is based on feedback mechanism, the source window gets the loss signal after some time which is determined by the queuing and propagation delay in its path. In our algorithm, this is determined beforehand from the signal received directly from the queue in the form of loss rate indicator variable λ. Whenever the λ value of any queue associated to any flow is nonzero, the algorithm switches the time step to minimum and records the window behavior for that short duration of time in which the loss event occurs. Also, before reaching to the source of that flow, the use of congestion signal for adjusting the time step is justified to minimize the possible error as less as possible. Adapting the time step to a suitable value as soon as possible helps tracking the network behavior accurately. Through this work, We find that the congestion signal for each flow is suitable variable for switching the time steps. We present various results which we have derived using our method and also compared with the regular fluid model solver simulation (FS) method without adapting time step length in our next section.

As described above, $dt_{\min}$ defines the minimum step length and its correct estimation is necessary. Since main aim of the network to see the fastest moving packets and record the interesting events, setting the minimum step length depends on highest link of possible bottlenecks. Following formula can be used to calculate minimum time



Start
Check the Loss at all the queues
Is there any Loss Event for any flow?
No
Yes
Set the step length to Minimum
Continue with the maximum step length
Congestion is over for all the flow?
No
Yes
Set the step length to maximum
Is simulation Over time over?
No
Yes
Finish

Fig. 4. Flow chart showing loss signal flow and decision making for step length adjustment.

step. A link C_i is congested if it satisfies following condition:

$$C_i < \sum_{l \in E} T(l,i)C_l + \sum_{k \in H} \frac{M_k n_k}{pk}, \tag{12}$$

and $dt_{\min}$ is given by

$$\min\left(\frac{1}{C_l}; l \in B\right), \tag{13}$$

where B is the set of bottleneck links which satisfy Eq. (12) and T is topology matrix, where $T(i,j) = 1$ if there exists at least one flow traversing queue j immediately after it traverses queue i. $H(i)$ is the set of TCP flows which have q_i as the first queue and p_k is the total propagation delay for flow k. Furthermore, the data flow from various level in the network and decision making for the step length adjustment is shown in Fig. 4.

4. Performance evaluation

All simulations were carried out on a workstation which has dual Xeon-3 GHz processors, 2 GB RAM on PC 2700

```
Define:
1    $dt$=Time Step Length for Simulation
2    $\lambda$=Loss Indication Rate
3    $RunTime$=Current network time in Simulator
4    $SimTime$=Time for which simulation to be run
5    $dt_(min)$ = FindDtMin() // Finds the best time step for the bottleneck link
Start:
     //to start with maximum Time Step Length
5    $dt = dt_{max}$ ;//to start with maximum Time Step Length
6    Repeat step 7 to step 10 for every $dt$ interval
     while(RunTime$\leq$SimTime)
7        If ( for any $\lambda ! = 0$)
             //Set Time Step Length to minimum for any loss event
8            set $dt = dt_{min}$
9        else
             //Set Time Step Length to maximum for no loss case
10           $dt = dt_{max}$

End:
```

Fig. 5. Adaptive time-step fluid simulation algorithm.

board. To present our results, we used a dumb-bell topology as shown in Fig. 6 to simulate a bottleneck link shared by two flows. Every link in this topology is equipped with a drop-tail queue with a maximum queue size of 500 packets. The packet size is kept fixed at 1000 B. The delay is 50 ms for bottleneck link and 10 ms for edge links.

4.1. Accuracy validation

Since our method is based on the existing fluid model solver (FS), we compare our time-adaptive fluid solver (AFS) with the normal fluid solver (FS) which does not have a time-adaptation mechanism. While solving using AFS, we start with the maximum value of time step which we set as

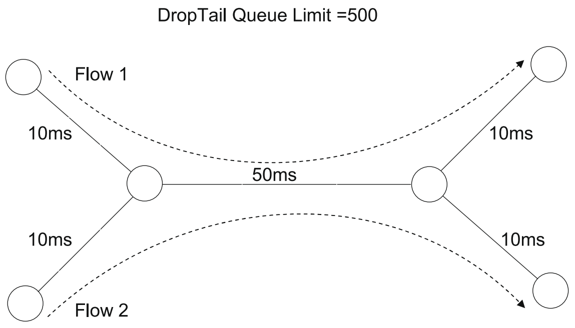


Fig. 6. Topology used for the simulation showing bottleneck link and two flows.

0.001 and suitably vary the minimum value keeping the minimum value for normal fluid simulation the same. In Fig. 7, the comparison of congestion window behavior between AFS and FS has been shown for 5 Gbps case. We observed that AFS and FS match good. Corresponding queue size behavior is presented in Fig. 8 which shows a good accuracy in spite of a very little mismatch.

We show how our accuracy depends on the change of step-size. In Fig. 9, it is shown that smaller the step length smaller the error. For the case of 10^{-3} we have higher error as compared to smaller step lengths.

4.2. Comparison

Fig. 10 shows the execution time comparison between the three simulation methods: NS2, FS and AFS. The NS2 simulation was not able to complete for the entire 20,000 s due to a limitation in the number of packets sent and hence we had to scale it. For these simulators to reach a congestion window limit, the simulations should be carried out for a longer time in the high-speed case. Since we ran the simulation for the bandwidth ranging from 100 Mbps to 10 Gbps with the network parameters that we have used, the 10 Gbps case shows its first loss event at around 17,000 s. Hence to have a fair comparison between these methods, the simulation should be carried out for 20,000 s. To accommodate all the values in the graph, a log scale has been used. The variation shows that FS achieves quite a good improvement over the packet simulator NS2. The execution time of NS2 increases exponentially as the bandwidth increases and overshoots in the

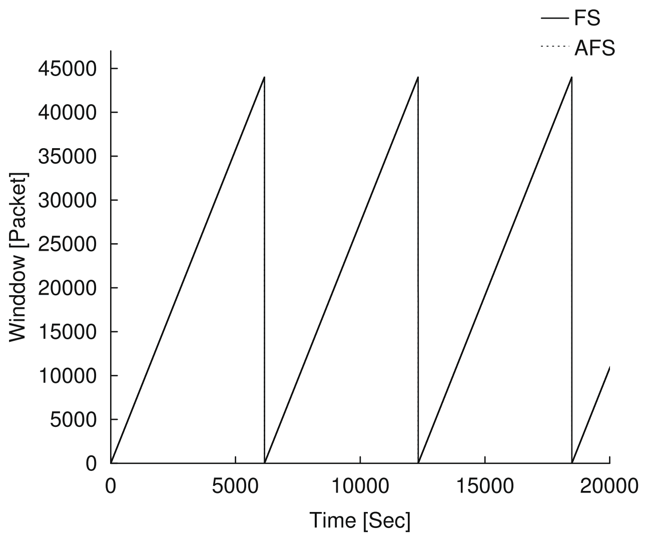


Fig. 7. Comparison of congestion window using AFS and fluid solver (FS) for 5 Gbps link.

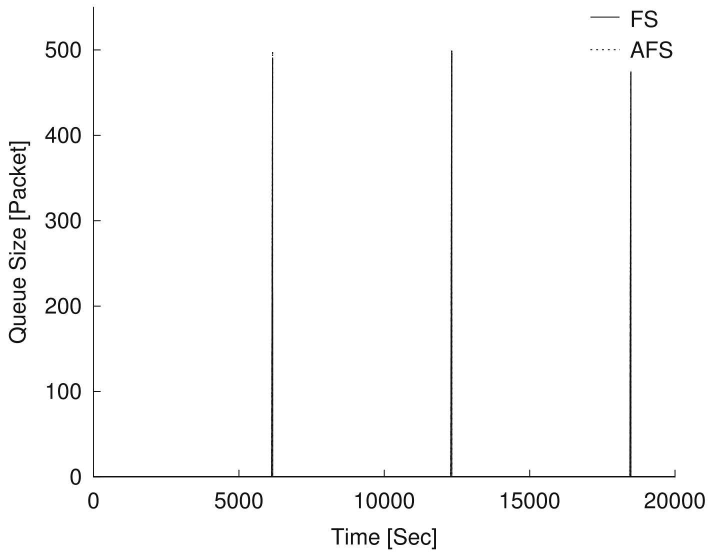


Fig. 8. Comparison of bottleneck queue using AFS and fluid solver (FS) 5 Gbps link.

case of 10 Gbps. In the case of traditional fluid simulator (FS) method, execution time increases because of smaller step length for the higher bandwidth case. As compared to the NS2 and FS method, our method achieves good improvements. Further, our simplified method uses fixed minimum and maximum step length and hence we observe a slight decrease in the execution time. This is because of lesser number of loss events for higher bandwidth in a given duration of time [22].

Fig. 10 shows the improvement through our method varies from 5 to 80 times as the bandwidth increases from 500 Mbps to 10 Gbps leaving NS2 far behind. In fluid model-based simulation, a cluster of closely-spaced packets is modeled as a fluid chunk at a specific time point. A fluid model-based simulator then keeps track of fluid chunks and their rate changes at each network component on the communication path from the source to the destination. As a large number of packets are abstracted as a single fluid chunk, the computational overhead is expected to be lessened. For microscopic analysis, lower time step means more accurate microscopic observation hence a desired time step would be the one which can capture microscopic behavior. In our case, one can easily observe TCP algorithm has two parts, increase of congestion window and decrease of congestion window. In protocol analysis one is more interested in the transitions between these two parts. If the algorithm is working either of two parts but not on the verge of transition, the behavior is easy to model and predict. The average behavior in these two parts can be modeled by putting more chunks or packets, i.e., decrease

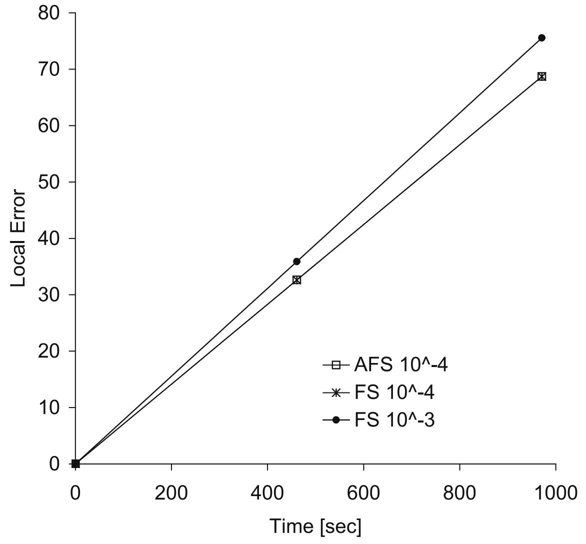


Fig. 9. Comparison of errors for different step length with respect to step length 10^{-6}.

of time step so that transition period is reached quickly. Hence, the execution time ignoring the coding overhead of FS method executing with a step length of T_{FS} is $O(1/t_{\mathrm{FS}})$ and for AFS method execution time is

$$O\left(\frac{t_{\mathrm{nl}}/T_{\mathrm{AFS}}^{\max}+t_l/T_{\mathrm{AFS}}^{\min}}{t_{\mathrm{ex}}}\right), \tag{14}$$

where $t_{\mathrm{nl}}, T_{\mathrm{AFS}}^{\max}, t_l, T_{\mathrm{AFS}}^{\min}$ and t_{ex} are execution time for no loss period, maximum time step, average time spend in loss period, corresponding time step length and total execution time for AFS method introduced in the paper respectively. Congestion time in high-speed network is a lot lesser than the no congestion time. Hence, the time improvement achieved in AFS method is very much dependent on the loss rate in the network which is very small in high-speed scenario.

In Fig. 11, we have shown the memory utilization for different methods. As we can see, the memory utilization is higher in the case of NS2 and still increases for FS too. The increment in non-swapped memory used by NS2 should be accounted for its inherent working mechanism. Since our system works on the delayed feedback mechanism, we have to store some variables to use it in later stage. In the case of TCP source using congestion information which reaches to it after one RTT, we observe increase in memory because the step-size is decreased, which forces more number of data to be stored. Whereas, in the case of AFS method, most of the time simulation is carried out with maximum step length, hence there is no significant increase in the utilization of non-swapped memory, hence results in better performance. However, for the complete characteristics of the network, the delayed feedback model is the best model to see the queue behavior and corresponding TCP algorithm behavior. The overhead incurred in this scenario is related to the data-structure maintained in the model to use it after some delay. In our case, we define it as average overhead incurred during the real network simulation time. Clearly, in the feedback system, the loss information arrives at the sender after one RTT (round trip time for worst case in which loss happens on sender's own queue). The message overhead is related to scaling of RTT w.r.t. time step. Hence, the loss information has to be saved in the system for 1 RTT. Therefore, by ignoring coding overhead, he loss information has to be saved for RTT/*time* − *step* time steps. Clearly, one can expect the message overhead for FS method is of the order of $(\mathrm{RTT}/T_{\mathrm{FS}})$ where T is the time-step-size of FS method. Whereas for AFS it is

$$O\left(\frac{t_{\mathrm{nl}}\mathrm{RTT}/T_{\mathrm{AFS}}^{\max}+t_l\mathrm{RTT}/T_{\mathrm{AFS}}^{\min}}{t_{\mathrm{ex}}}\right). \tag{15}$$

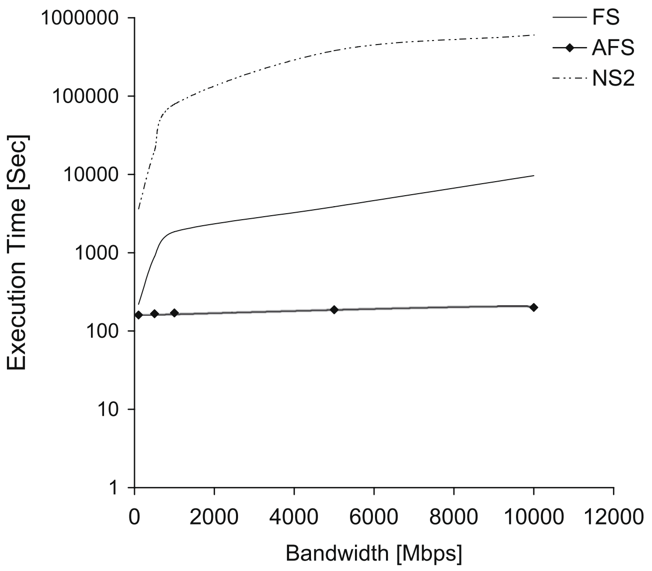


Fig. 10. Comparison of execution time for NS2, FS and AFS methods with the variation of bandwidth.

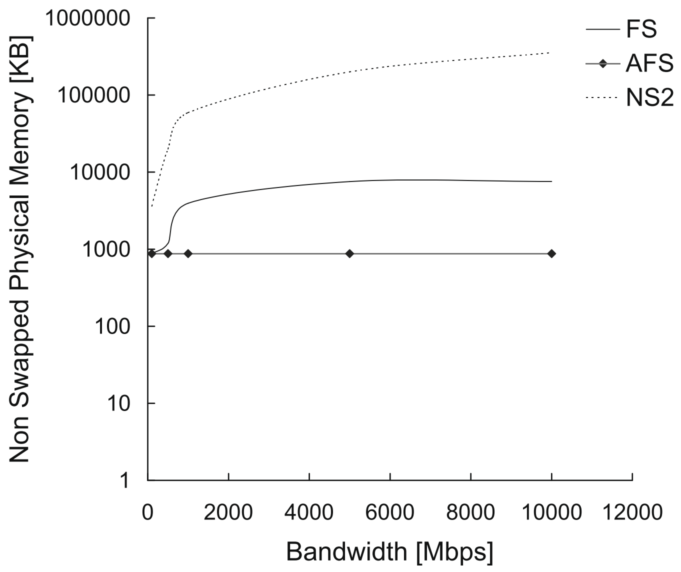


Fig. 11. Comparison of non-swap memory utilization for NS2, FS and AFS with the variation of bandwidth.

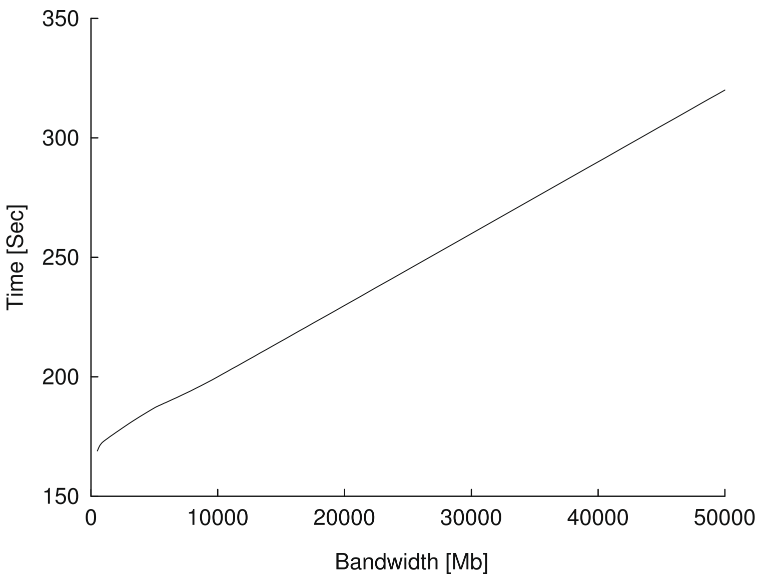


Fig. 12. Execution time variation of AFS method with bandwidth.

Average time spent in the congestion period is a lot lesser than the minimum time step $T_{\mathrm{AFS}}^{\mathrm{min}}$. Hence, we can observe that average message overhead of adaptive time step simulation (AFS) method is directly related to time step and RTT of the network.

4.3. Scalability

In Fig. 12, we have shown the variation of execution time till the bandwidth of 50 Gbps. As bandwidth increases, the AFS method introduced in this paper shows good results. The increment of the time corresponds to variation of different minimum step length. As we go higher, the minimum time step needed to simulate decreases and hence the adaptive decrease part takes more number of steps giving a slightly increment in execution time. Therefore, the method introduced in this paper is scalable to bandwidth of modern high-speed networks.

4.4. Effect of loss

The bandwidth is 10 Gbps and other input parameters are same as discussed above. As shown in Fig. 13, the execution times variation of AFS and FS approach have been

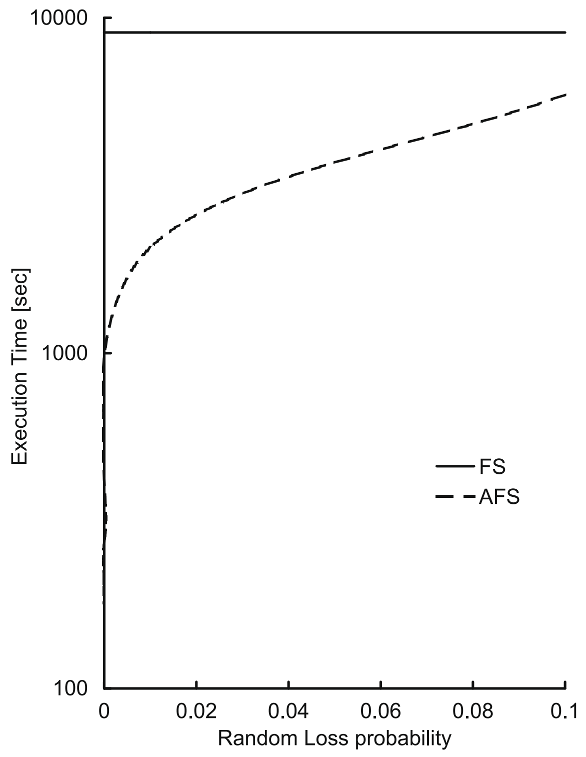


Fig. 13. Effect of random loss on AFS and FS execution time.

compared with for different random packet loss probability. While execution time of FS method remains almost constant at a higher value, the AFS execution time for AFS method varies. Initially, when the value of random packet loss probability is low, the queue loss is dominating factor and has the major number of loss events (which is still less) giving the low value of execution time. As we increase the random packet loss probability, it predominantly accounts for all the packet losses in the network and the queue loss becomes negligible accounting for the increased execution time. Since the more number of packet loss makes AFS execution longer in minimum time step region, it is also worth mentioning that as the packet loss increases the AFS execution time also increases. Hence, a good improvement is achieved in high-speed and low-loss scenario (which is predominantly the high-speed scenario in consideration) over the simple fluid-based approach.

5. A simple flow level simulation of high-speed network transfer control protocol

In this section, performance of different high-speed TCP variants is presented for a simple configuration. We use our AFS algorithm method for the evaluation. Our methodologies involve mainly AIMD algorithm for TCP as we presented in previous sections. However, the proposed framework is valid for any TCP variants for which we can develop a time continuous model. This section is focused on the behavior of different TCP variants. First, we present fluid models of the different high-speed TCP variants by giving model equations for each and then we evaluate their performances for some basic configurations.

5.1. Background and methodology

TCP congestion control is made of probing phase and decreasing phase. The probing phase of a TCP consists of an exponential increase phase (i.e., the slow start phase) and a linear increase phase (congestion avoidance phase). The probing phase stops when congestion occurs in the network and at this point TCP starts decrease phase. Almost all the TCP variants have to go through these phases. We can generalize the linear increase and decrease of window by representing them with ordinary differential equations of the form:

$$\frac{dW_i(t)}{dt} = \frac{a(W_i(t) < M_i)}{R_i(t)} - W_i(t)b\lambda_i(t), \tag{16}$$

where a and b are defined as increment and decrement factors, respectively. The differential equations govern the flow level behavior of the network. Main focus of this section is on evaluation of congestion control algorithms, therefore slow start, ECN mechanism, etc., are not taken into account. We give the functions for these parameters for different TCP variants in the tabular form (Table 1).

To evaluate these protocols, two scenarios are created: (A) Single-flow scenario is focused on how different TCP congestion control algorithms behave with respect to drop-tail queuing provisioning. (B) Two-flow scenario is where two TCP flows share the same bottleneck link. We observe convergence properties for intra-protocol and TCP friendliness and fairness is left for further evaluation, where high-speed TCP variants compete with a conventional TCP. Traditionally, router buffers are provisioned according to the delay-bandwidth product (BDP) rule:

Table 1
Listing increment and decrement parameter of high-speed TCP variants.

TCP variant	a	b
TCP-Reno	1	0.5
STCP	$0.01w$	0.125
HSTCP	$2\frac{w^{0.8}b}{2-b}$	$(0.1-0.5)\frac{\log(w)-\log(w_{\text{low}})}{\log(w_{\text{high}})-\log(w_{\text{low}})}$ $+0.5$
CUBIC TCP	$\text{Min}(target_w - w, S_{\max}R)$, where $target_w = origin\_point + c(\Delta_{\text{th}} - K)^3$ $K = (b.prevMax_w/c)^{\frac{1}{3}}$	0.2
H-TCP	$1 + 10(\Delta_i - \Delta_{\text{th}}) + \left(\frac{\Delta_i - \Delta_{th}}{2}\right)^2$	$1 - \frac{R_{\min}}{R_{\max}}$
FAST TCP	$\text{Min}(w, \gamma(2baseR) - \text{avgRTT})\frac{w}{\text{RTT}} + \alpha$	0.5

namely, one chooses the buffer size as $q = B\text{X}\text{RTT}$, where B is the rate of the link served by the router, and RTT is the typical round trip time (RTT) experienced by connections utilizing the buffer. This amount of buffering allows 100% link utilization. However, last few years, buffer sizing of router attracted lots of attention. Having small buffers is beneficial in terms of amount of memory, required physical space, energy consumption, and price of the router. In our studies, the main advantage of having small buffers is the reduction in queuing delays and jitter. Hence we set all the buffer sizes as 500 packets. The end to end delay is set to 60 ms, i.e., 120 ms round trip time, when there is no queue delay.

5.2. Single-flow scenario

The goal is to analyze how different TCP congestion control algorithms behave with respect to drop-tail queuing discipline. We perform a simple set of simulations using a single source and single receiver with an average bottleneck buffer size of 500 packets.

5.2.1. Drop-tail queue

Drop tail is a simple queue management algorithm used by Internet routers to decide when to drop packets. With tail drop, when the queue is filled to its maximum capacity, the newly arriving packets are dropped until the queue has enough room to accept incoming traffic. Once a queue has been filled, the router begins discarding all additional datagrams, thus dropping the tail of the sequence of datagrams. Drop tail is an interesting queuing discipline where all the packets are dropped once queue limit is reached and in high-speed network. This behavior can be severe because the protocol is operating at a maximum sending rate causing burstiness which eventually led to multiple losses. In this section, we observe the behavior the different TCP variants to show the queue occupancy for a drop-tail queue with a buffer size of 500 packets. We also observe time taken by a particular flow to reach the bottleneck limit.

HSTCP: High-speed response function behaves as an aggregate of N TCP connection. The response function gives a straight line on a log–log scale. In Figs. 14 and 15, we show HSTCP window evolution and queuing occupancy. We observe that, HSTCP takes 370 s to reach the bottleneck link. At that point, a sending rate is very high and it leads to multiple packet losses. The queue is oscillating with the same time period. Interestingly, high speed TCP congestion control algorithm working at a very high sending rate overflows queues for a very short period of time. This aggressiveness is the price paid by high-speed TCP variants for their increased scalability. After the congestion occurs, the window increase is slow initially; and as time passes, it gets steeper. For this time period, the average window size is 53,400 which is approximately 1/3 of the peak window size.

CUBIC: CUBIC TCP uses a cubic function whose shape is similar to BIC TCP. This TCP comes into the picture carrying all the features of BIC TCP but friendlier than BIC TCP. Figs. 16 and 17 show CUBIC window behavior and queue occupancy respectively. In the case of CUBIC, the window grows very fast; but as it gets closer to $W_{\max}$ (the previous congestion window just before congestion), it slows down its growth. At $W_{\max}$, its increment

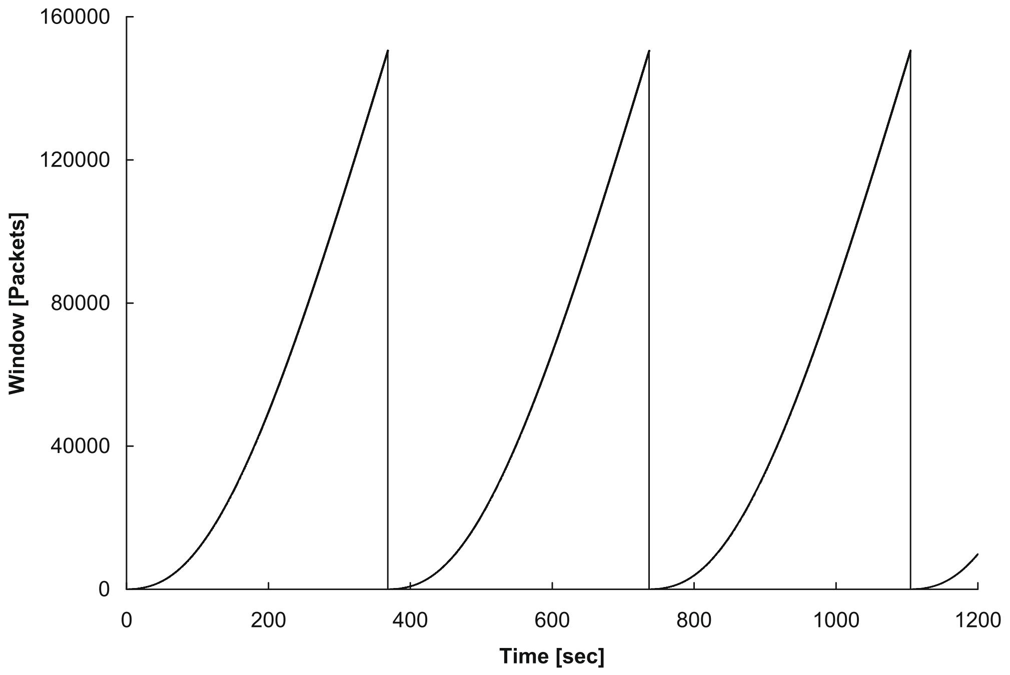


Fig. 14. Congestion window for HSTCP.

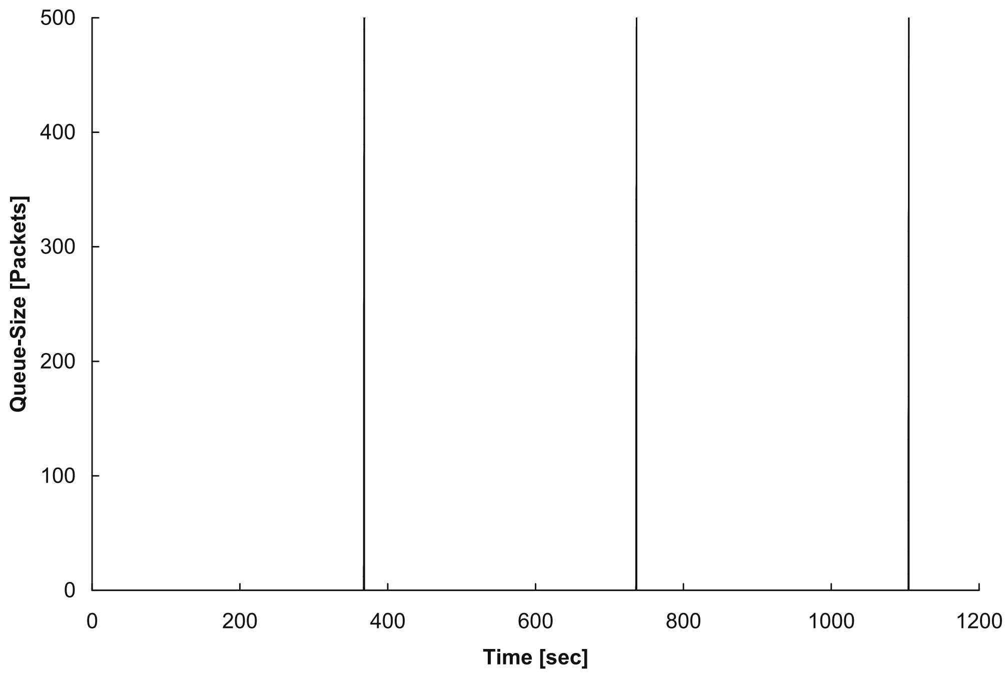


Fig. 15. Queue for HSTCP.

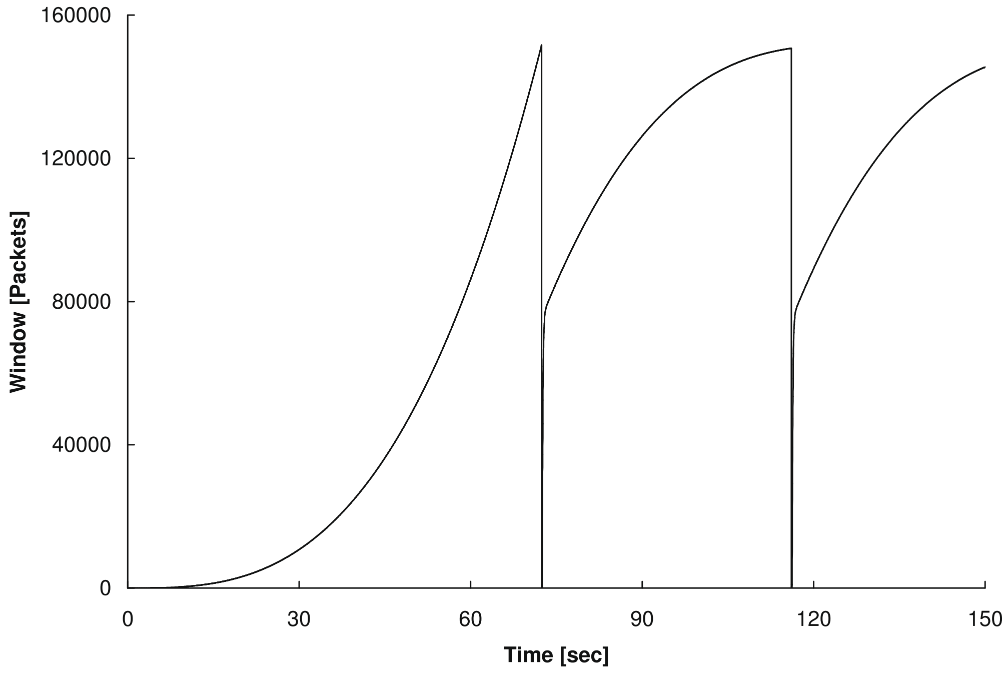


Fig. 16. Congestion window for the CUBIC TCP.

becomes zero. Since we considered ideal queue behavior with no traffic abnormality, the behavior past $W_{\max}$ is not visible. The congestion window oscillates with a period of 44 s. As we can see, irrespective of queue size behavior, it paces down the sending rate when close to $W_{\max}$ and hence sending packets nearly at the same rate of bottleneck link. At this point queue don't get empty for 3 s. The increment in congestion window continues for 3 s passing the bottleneck service rate and hence causing congestion. At this point, congestion window drops and rises fast. The average congestion window for the oscillation time period is 126,500 giving 83% link utilization where as average queue size is 17 packets. Another interesting observation is that queue is busier than that of HSTCP. However, drop-tail queuing discipline does not decrease link utilization much.

STCP: STCP uses multiple increase and multiple decrease methodology for congestion control. In Figs. 18 and 19, window and queue occupancy behavior is shown respectively. As we can see when the congestion window is small the increase parameter employs slow window increment and when high it becomes very aggressive. This aggressiveness behavior although provides scalability, it suffers through heavy packet losses.

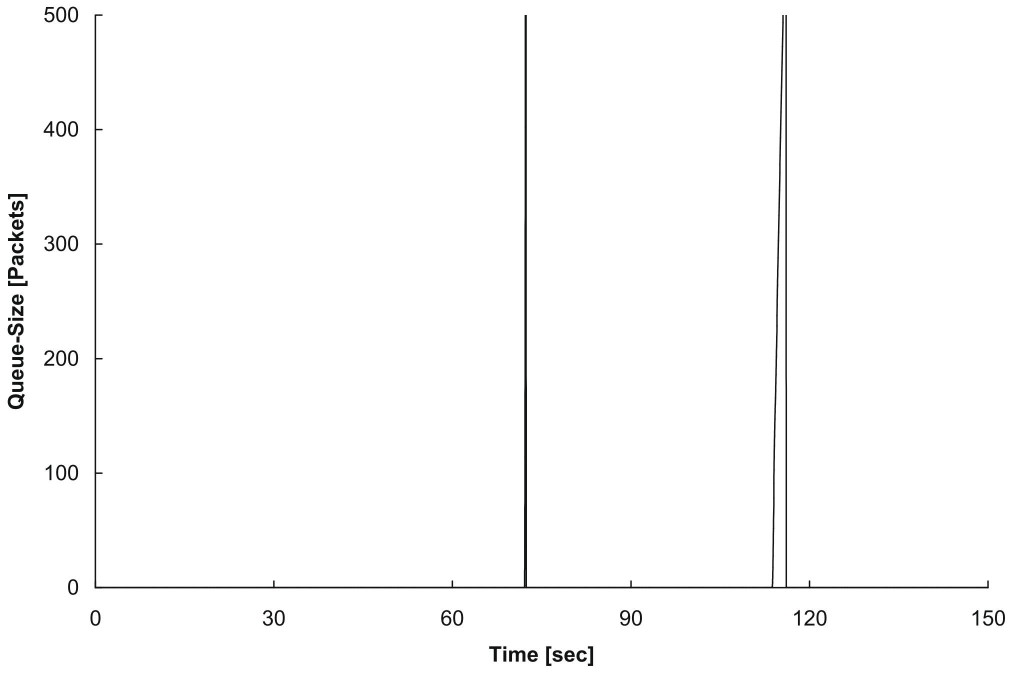


Fig. 17. Queue for CUBIC TCP.

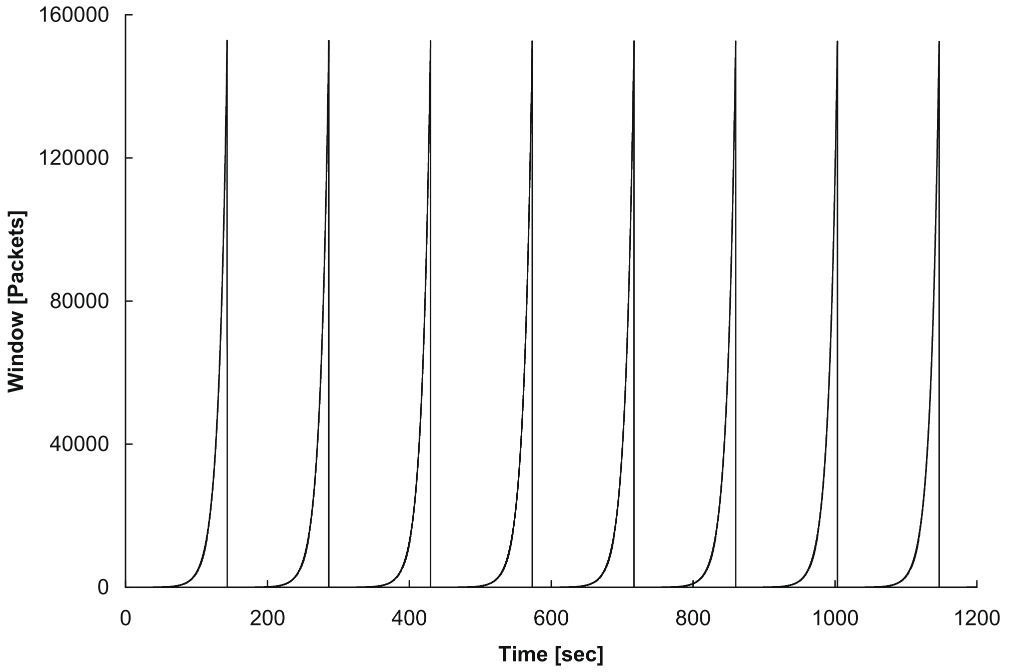


Fig. 18. Congestion window for STCP.

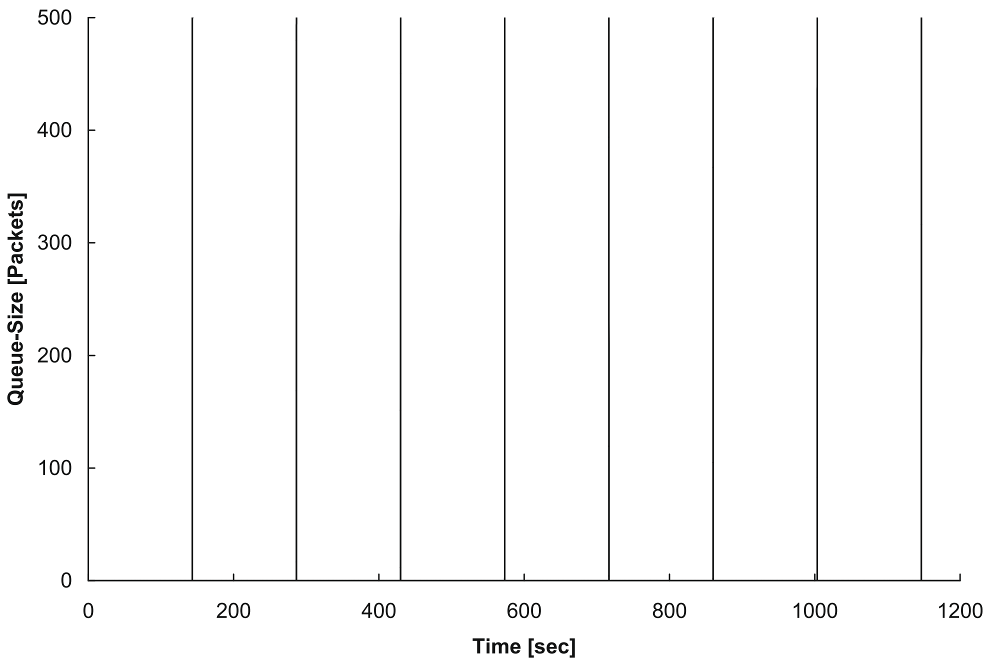


Fig. 19. Queue for STCP.

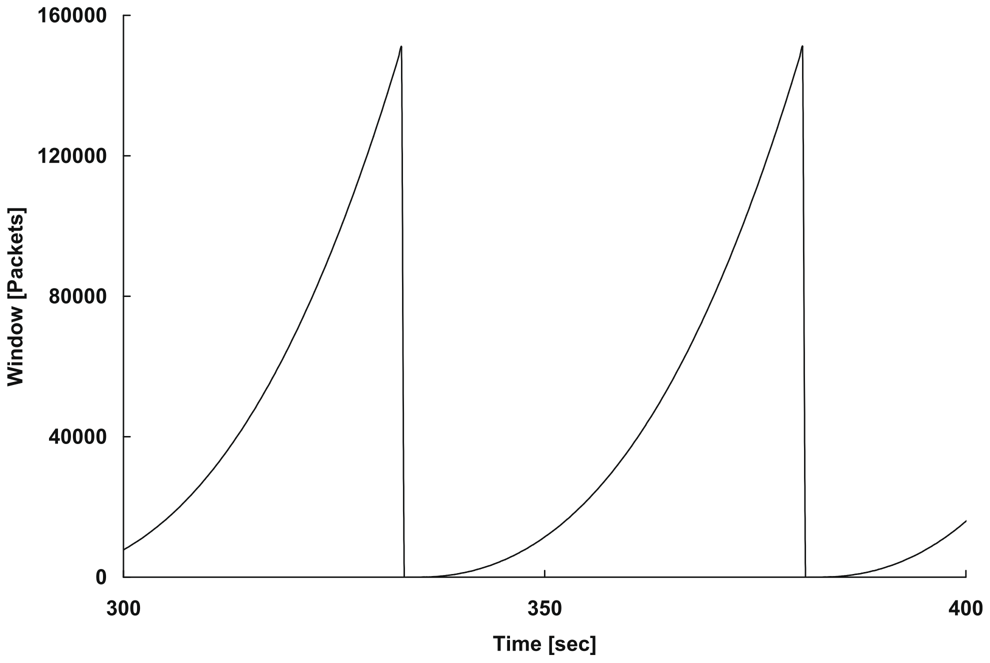


Fig. 20. Congestion window for HTCP.

In figure, we observe that window size oscillates with a time period of 140 s. Average window size during this time period is 12,829 which is less than 10% of peak bottleneck link. At this peak value, drop-tail router drops all the packets and STCP congestion window drops drastically. However, the queue momentarily gets full and after the congestion period is over, it empties instantly attributing to high link service rate. STCP performs very poorly in the case of drop-tail queuing discipline.

HTCP: HTCP is suggested as a modification to conventional TCP. Additive increase factor is a quadratic function of time since last congestion and decrease factor is ratio of minimum RTT observed and maximum RTT observed. In Figs. 20 and 21, we show congestion window behavior and queue occupancy respectively. One quick observation tells us that even HTCP suffers through burstiness behavior causing multiple losses. The congestion window oscillates with a time period of 47 s. After HTCP recovers from congestion the additive increase function attributes to slow increase in the beginning and as time increases since the last congestion it aggressively utilizes the link bandwidth. Although this aggressiveness enables protocol to reach

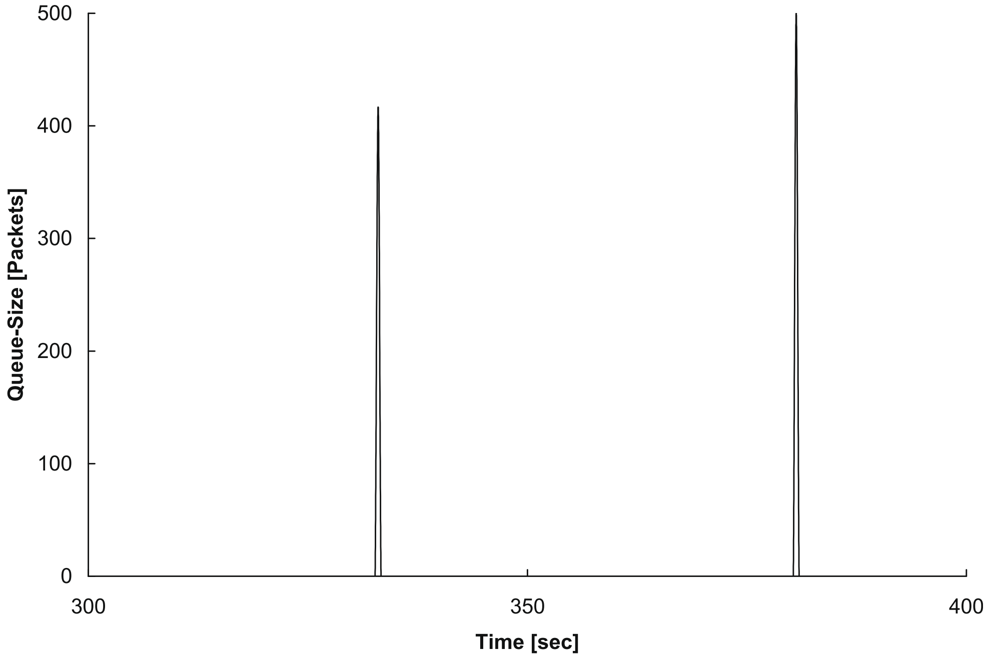


Fig. 21. Queue for HTCP.

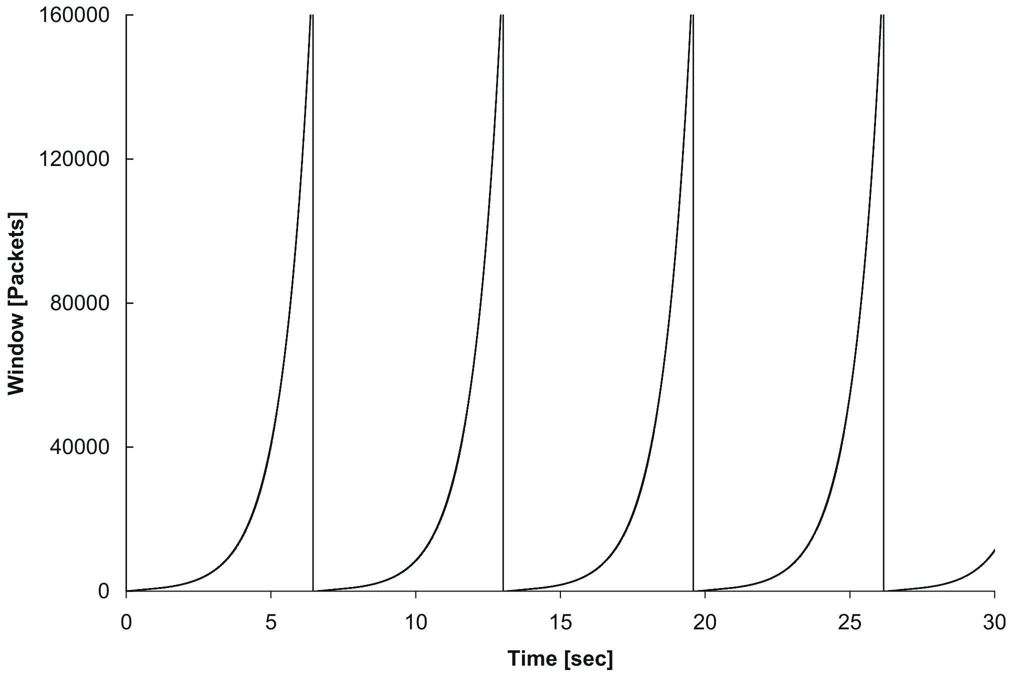


Fig. 22. Congestion window for FAST.

the next congestion quickly, drop-tail router causes a lots of loss that it drops drastically. Average congestion window value in this case is 44,000 which is 30% of bottleneck link.

FAST: FAST TCP is an equation based protocol and has been introduced to control the stability properties of dynamic congestion window behavior. In Figs. 22 and 23, we show congestion window and queue occupancy behavior of FAST TCP respectively. In figure, we observe even FAST TCP suffers through multiple losses and on congestion, congestion window decreases dramatically. α (= Number data packets that can be kept in bottleneck queue) decides this aggressiveness behavior. In this case, we set alpha parameter in excess because FAST does not have any method to know this parameter. If the value of this parameter is large, it can degrade overall throughput. This is an obvious assumption however, it is not true for small buffer queues. Queue size slowly increases and after congestion empties by service rate of bottleneck link. FAST TCP oscillates with a time

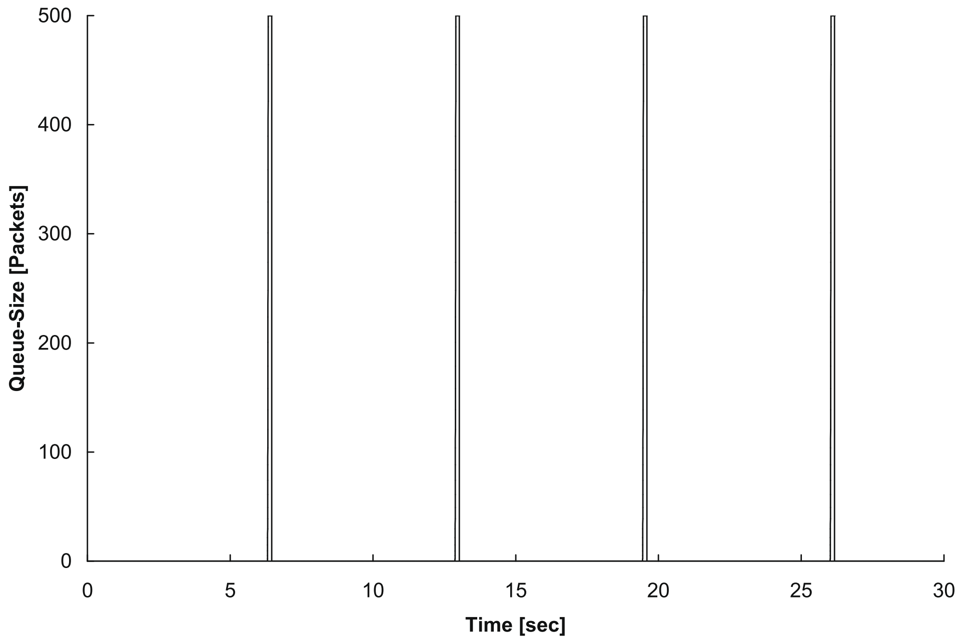


Fig. 23. Queue for FAST.

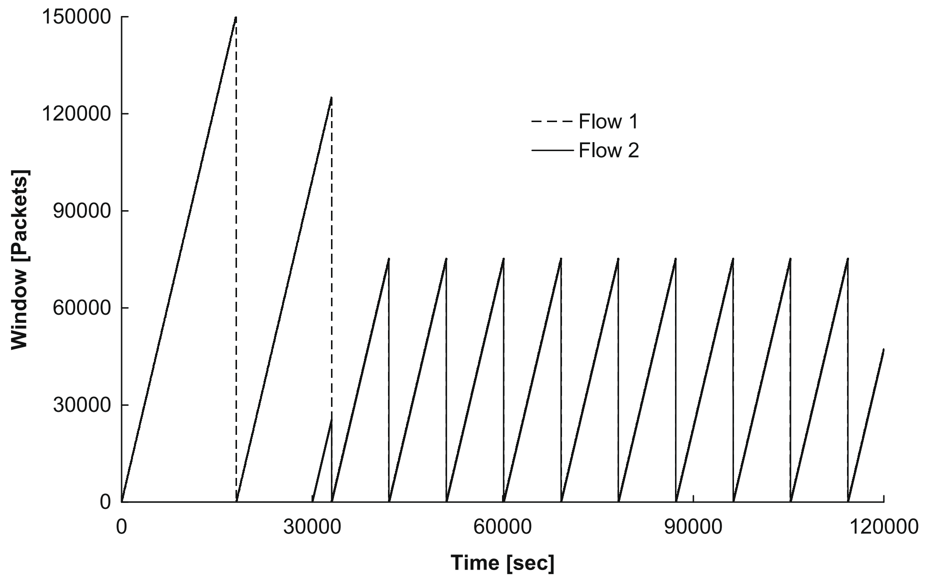


Fig. 24. Congestion window for TCP-Reno intra-protocol convergence.

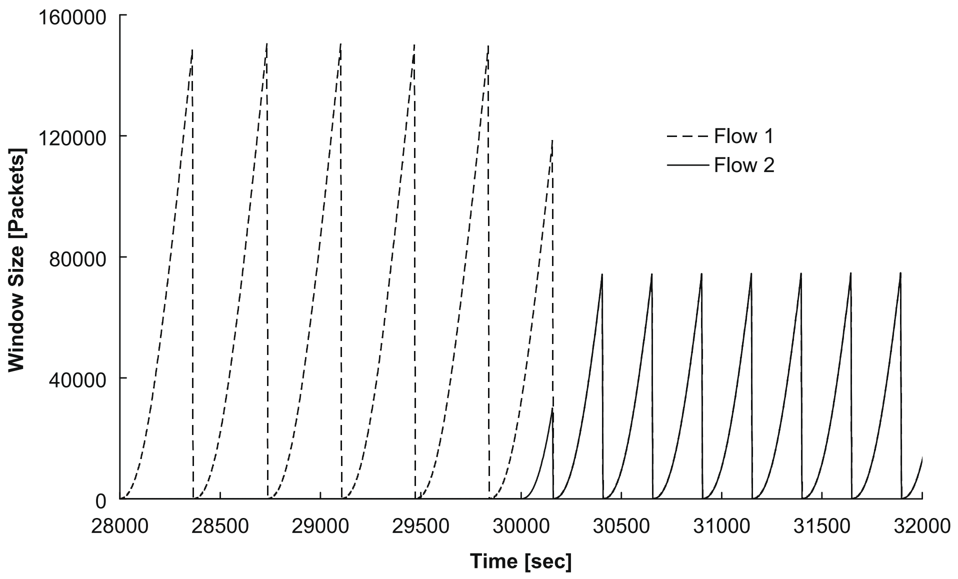


Fig. 25. Congestion window for HSTCP intra-protocol convergence.

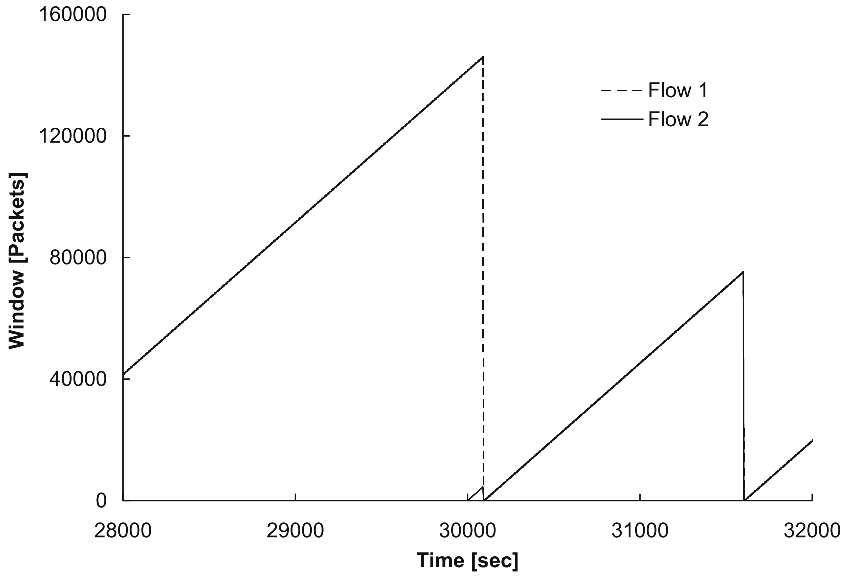


Fig. 26. Congestion window for CUBIC TCP intra-protocol convergence.

period of 6.5 s. Since FAST TCP tries to reach the maximum sending rate as fast as it can,[3] if the update period of the congestion window is decreased, this behavior would be more aggressive because the update has to be done more frequently. Average congestion window is 27,000, which is less than 20% of link utilization. As we can see most part of the oscillating period of FAST TCP, it has smaller window and becomes aggressive when around congestion.

[3] The update period is set to 1 s, which gives the average rate over 1 s interval instead of RTT.

5.3. Two-flow scenario

5.3.1. Convergence

We show figures for convergence properties for different high-speed TCP variants.

TCP-Reno has slow convergence which is seen in Fig. 24. Convergence is achieved when the sum of two flows exceeds the bottleneck link capacity. This slow convergence is also attributed to the time when the other flow starts, we can expect fast convergence when one flow is close to the bottleneck link capacity and other is just starting. The convergence is achieved because of drastic decrease in congestion window because of high losses.

Fig. 25 shows HS-TCP cwnd time history of flows with the same round-trip time following startup of a second flow. It can be seen that the flows do converge to fairness, but the convergence time can be long. This effect can become more pronounced as the path propagation delay is increased (to be explored later). Recall that the AIMD increase parameters are functions of cwnd in HS-TCP. The slow convergence appears to originate in the asymmetry that exists in HS-TCP between the AIMD parameters of newly started flows (with small cwnd) and existing flows (with large cwnd).

CUBIC TCP converges a lot faster (Fig. 26) than HS-TCP. CUBIC TCP employs a cubic function which is concave in nature. Therefore, after a window reduction, the window grows very fast and as it gets closer to link capacity, it slows down its growth. At that point, the other flow tries to find the upper bound of maximum growth. At the upper bound of link capacity, the old flow's increment rate becomes zero. After that, the congestion window grows slowly and accelerates its growth as it moves away from bottleneck link capacity giving more chance to the other flow to catch up and achieve a fair share.

In Fig. 27, we show a typical example of measured congestion window for scalable-TCP. Generally, the congestion windows either do not converge to fairness or converge very slowly (not reaching fairness within the 10-min dura-

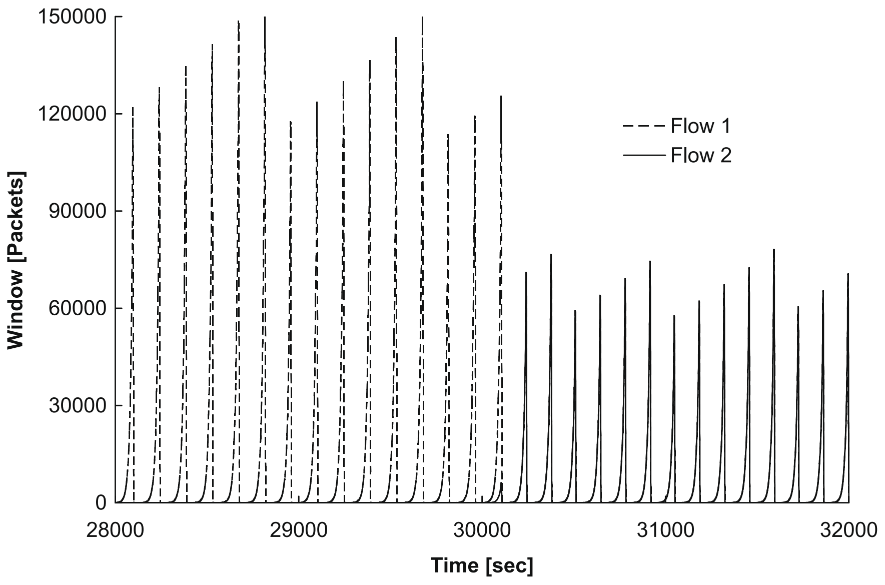


Fig. 27. Congestion window for STCP intra-protocol convergence.

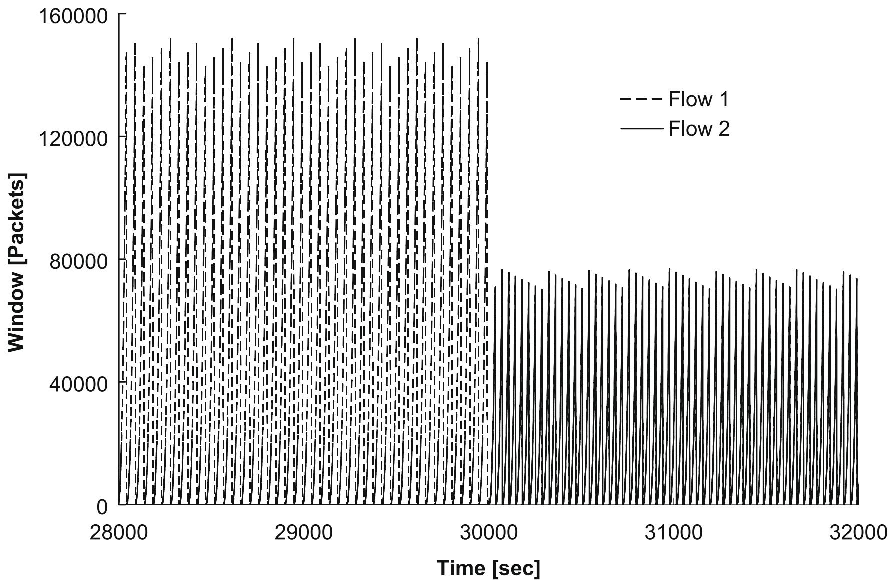


Fig. 28. Congestion window for HTCP intra-protocol convergence.

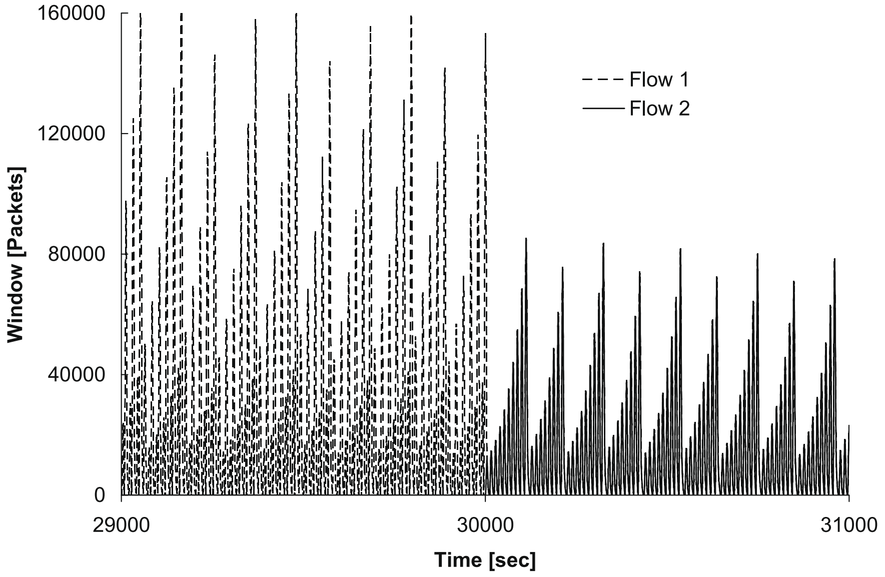


Fig. 29. Congestion window for FAST TCP intra-protocol convergence.

tion of these tests). However, high-speed network differs from slow or traditional networks in the sense that, it can undergo multiple losses. At flow level the multiple losses causes drastic reduction of congestion. Sometimes it results into slow start. Hence, because of high losses, it is expected that STCP flows can converge aggressively as seen in our work.

H-TCP flows show rapid convergence (Fig. 28). Increase function of H-TCP is a function of the time since last congestion, hence older flow has not experienced congestion and having faster increase function than the newer flows. This paradigm causes faster flows to drop more when the new flows join and this causes congestion events. After the congestion event, both exhibit the same increment function (as time) since last congestion is same for both. Therefore, with the same increment and decrement parameters H-TCP shows the same convergence properties in terms of number of congestion epochs. In the case of H-TCP, congestion epochs occur quicker than two TCP flows working together, faster convergence is expected than that of TCP.

In Fig. 29, we show as new flow joins, FAST TCP converges to the new equilibrium rate rapidly and stably. Window converges exponentially to the equilibrium at a different rate that depends on queuing delay. Both the flows try to keep the maximum number of packets in the queue which depends on variable alpha which is a configurable parameter. We can expect the change of convergence behavior as we change this parameter.

6. Conclusions

We recognize that fluid-based simulation is an effective method for simulation of high-speed networks. Our contribution of this work is to provide a scalable fluid-based simulation method. Although the fluid-based simulation method has a significant reduction in terms of computational time, it still suffers a scalability problem for the networks with bandwidth greater than 10 Gbps. Since the fluid-based simulation method uses a constant time step to numerically solve the system of differential equations, it needs to decrease the size of time step in case of a larger bandwidth. The decrease of time step produces more number of time steps, which induce more amount of computational cost.

We have developed the time-adaptive method for the numerical solver for a system of differential equations to reduce the computational cost. The proposed method adjusts the time-step-size for the numerical solver in order to reduce the computational cost while maintaining the accuracy of simulation results. The time-adaptive method uses a larger time step for the part of linear increase and a smaller time step for the part of multiplicative decrease in the event of packet loss. Since the event of packet loss is synchronized with the event of multiplicative decrease, we can adjust the time step based on the event of packet loss and determination of time step is based on congested link in the network.

Comparisons between the time-adaptive method and the constant time step method show that the proposed method significantly reduces the computational cost while maintaining the level of accuracy compared to the constant time step method.

While the TCP variants studied are all successful at improving the link utilization in a relatively static environment with long-lived flows, in our tests many of the variants exhibit poor responsiveness to changing network conditions. We observe that scalable-TCP and HS-TCP can suffer from extremely slow convergence times following the startup of a new flow. It is important that we evaluate the high-speed TCPs in various other different environment. Therefore, more studies have to be done to evaluate these protocols for use in high-speed networks. Simulation study of high-speed network transfer protocol is another goal of our future work.

Acknowledgements

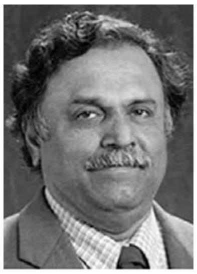
This work was supported in part by the NSF MRI grant (#0821741) and by Department of Defense (Office of Naval Research) grant (DEPSCoR).

References

[1] NLR. <http://www.nlr.net/>.
[2] LONI. <http://www.cct.lsu.edu/news/loniforum.php/>.
[3] C. Jin, D.X. Wei, S.H. Low, Fast TCP: motivation, architecture, algorithms, performance, in: Proceedings of the Conference on Computer Communications (IEEE INFOCOM), Hong Kong, March 2004.
[4] Sally Floyd, Highspeed TCP for large congestion windows, Tech. Rep. RFC 3649, December 2003.
[5] Tom Kelly, Scalable TCP: improving performance in highspeed wide area networks, ACM SIGCOMM Computer Communication Review 33 (2) (2003) 83–91.
[6] L. Xu, K.Harfoush, I. Rhee, Binary increase congestion control (BIC) for fast long-distance networks, in: IEEE INFOCOM, March 2004.
[7] D.J. Leith, R.N. Shorten, H-TCP: TCP for high-speed and long-distance networks, in: Proceedings of PFLDnet, 2004.
[8] NS2. <http://www.isi.edu/nsnam/ns>.
[9] Opnet. <http://www.opnet.com/>.
[10] NS2 LargeSim. <http://www.isi.edu/nsnam/ns/ns-largesim.html>.
[11] SSFnet. <http://www.ssfnet.org>.
[12] NistNet. <http://snad.ncsl.nist.gov/itg/nistnet/>.
[13] Y. Liu, F.L. Presti, Y. Liu, F.L. Presti, V. Misra, D. Towsley, Y. Gu, Fluid models and solutions for large-scale ip networks, in: ACM SIGMETRICS, 2003.
[14] Yixin Wu, Suman Kumar, Seung-Jong Park, On transport protocol performance measurement over 10 Gbps high speed optical network, in: Proceedings of ICCCN, 2009.
[15] S.H. Low, D.E. Lapsley, Optimization flow control. I: Basic algorithm and convergence, IEEE/ACM Transactions on Networking 7 (6) (1999) 861–875.
[16] H.Q. Ye, Stability of data networks under an optimization-based bandwidth allocation, IEEE/ACM Transactions on Automatic Control 48 (7) (2003) 1238–1242.
[17] Vishal Misra, WeiBo Gong, Don Towsley, A fluid-based analysis of a network of AQM routers supporting TCP flows with an application to red, in: ACM SIGCOMM, September 2000.
[18] F. Baccelli, D. Hong, Flow level simulation of large ip networks, in: INFOCOM, 2003.
[19] Anlu Yan, Wei-Bo Gong, Time-driven fluid simulation for high-speed networks, IEEE Transactions on Information Theory 45 (5) (1999).
[20] Arieh Iserles, A First Course in the Numerical Analysis of Differential Equations, Cambridge University Press, 1996.
[21] DWDM. <http://www.iec.org/online/tutorials/dwdm/>.
[22] Yaaser Mohamed, Suman Kumar, Seung-Jong Park, Simulation results of variants of TCP over 10 Gbps high speed optical networks, Technical Report of Computer Science and CCT at Louisiana State University, 2005.

Suman Kumar is a Ph.D. candidate in the Computer Science Department and Center for Computation Technology at Louisiana State University. He received his B.Tech. degree in Electronics and Communication Engineering from the Indian Institute of Technology (IIT), BHU, India in 2003. During his B.Tech., he published a few research papers in international journals and conferences in solid state and microwave devices. His current research interests include high-speed networks, wireless sensor networks, network modelling and simulation, high-performance computing, and algorithm optimization.

Seung-Jong Park is an assistant professor in the Computer Science Department and Center for Computation Technology at Louisiana State University. He received his Ph.D. from The School of Electrical and Computer Engineering at Georgia Institute of Technology, 2004. Prior to that, he had also received a B.S. degree in Computer Science at Korea University, Seoul, Korea and a M.S. degree in Computer Science from KAIST (Korea Advanced Institute of Science and Technology), Teajon, Korea in 1993 and 1995, respectively. His research interests are in protocols over wireless networks and high-speed optical networks and mobile computing.



S. Sitharama Iyengar is the Chairman and Roy Paul Daniels Chaired Professor of Computer Science at Louisiana State University, Baton Rouge, LA and is also the Satish Dhawan Chaired Professor at the Indian Institute of Science, Bangalore. His publications include 6 textbooks and over 380 research papers. His research interests include high-performance algorithms, data-structures, sensor fusion, data mining, and intelligent systems. His research has been funded by the NSF, ONR, NASA, DoE/ORNL, US Army Research Office, and the DARPA and MURI Programs. He is a fellow of IEEE, ACM, AAAS, and SDPS. He is a recipient of IEEE awards, best paper awards, the Distinguished Alumnus award of the Indian Institute of Science, Bangalore, and other awards. He has served as the editor of several IEEE journals and is the founding editor-in-chief of the International Journal of Distributed Sensor Networks.